**slice:**

We create a sublist of a list by specifying a range of indices.

Semantically specifying the slice is similar to that of range function.

But syntactically, it is different.

The result of slicing is a new list.

The examples are self explanatory.

# file 1\_slice.py

# indices:

# 0 1 2 3 4 5 6

b = [12, 23, 34, 45, 56, 67, 78]

print(b[2:5]) # [34, 45, 56]

print(b[:5]) # b[0:5] # [12, 23, 34, 45, 56]

print(b[2:]) # b[2:len(b)] # [34, 45, 56, 67, 78]

print(b[2:6:2]) # init : 2, final value one past the end : 6; step 2 # [34, 56]

print(b[::2]) # init : 0, final value one past the end : len(list); step : 2

# [12, 34, 56, 78]

print(b[::-1]) # reverse the elements of the list # [78, 67, 56, 45, 34, 23, 12]

print("what : ", b[:5:-1]) # what : [78]

Let us try a couple of examples based on slicing of lists.

**Example: find the biggest**

# file : 2\_biggest.py

# find the biggest

# algorithm:

# assume the first element in index 0 as the biggest

# walk thro the remaining elements of the list.

# compare and update big if necessary.

# output biggest

a = [22, 44, 11, 55, 33]

big = a[0]

for e in **a[1:]:** # observe : all elements but for the 0th element

if e > big :

big = e

print("biggest : ", big)

**Example: find the total of a slice**

# file : 3\_find\_total.py

# find the total of a batsman

# list contains the name of the batsman and his scores in # of innings

# find the total score of Kohli

scores = [ "kohli", 0, 82, 25, 120, 76]

total = 0

for e in **scores[1:]:**

total += e

print("total : ", total)

$ python 3\_find\_total.py

total : 303

**assignment of slice:**

**When we use to the left of assignment,**

**a) all the elements in that slice will be removed**

**b) all the elements on the right will replace them**

Note :

**1. The right hand side should signify # of elements. It should be an iterable.**

**2. Number of elements on either side need not be same.**

Observe the outputs and comments after each assignment.

# file : 4\_slice\_assignment.py

# assignment of slice

a = [ 10, 20, 30, 40, 50, 60]

# remove the elements on the left

# replace by the elements on the right

a[2:4] = [100, 200] #right hand side(rhs) should be iterable

print(a)

# [10, 20, 100, 200, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]

a[2:4] = [1000, 2000, 3000, 4000]

print(a) # list has become bigger

# [10, 20, 1000, 2000, 3000, 4000, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]

a[2:4] = [] # list has become smaller

print(a)

# [10, 20, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]

a[2:4] = "pesu" # str is iterable

print(a)

# [10, 20, 'p', 'e', 's', 'u', 50, 60]

a = [ 10, 20, 30, 40, 50, 60]

a[2] = "fool" # a[2] is not a slice;

print(a)

#[10, 20, 'fool', 40, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]

a[2:3] = "fool" #a[2:3] is a slice

print(a)

# [10, 20, 'f', 'o', 'o', 'l', 40, 50, 60]

**List of lists:**

Some of a list themselves are a list.

If list is like a vector of Math, list of lists is like a matrix.

a = [

[11, 22, 33],

[44, 55, 66]

]

This is an example of a list of lists.

print(len(a), len(a[1]) # 2 3

a is a list of 2 elements - a[1] is a list of 3 elements.

The list of lists need not be rectangular.

b = [

[ 1, 2, 3 ]

[ 4, 5, 6, 7],

[ 8, 9]

]

Let us try some simple examples of lists of lists.

**example 1:**

**Generate an identity matrix.**

version 1:

This creates an empty list. Adds n empty rows. Appends an element each time

in the innermost loop. The element is generated by the expression (i//j) \* (j//i).

This expression will be 1 if i = j and 0 otherwise.

This is based on the trick of integer division - not a good idea.

# file : 5\_list\_identity.py

"""

# generates identity matrix

# bad program - depending on integer division

# version 1:

"""

n = 4

a = [] # empty matrix

for i in range(1, n + 1) :

a.append([]) # add a row each time

for j in range(1, n + 1) :

a[i-1].append((i//j) \* (j//i)) # trick ?

for x in a :

for e in x :

print(e, end = " ")

print()

version 2:

We create the list of lists the way we did last time. But we append the element

based on whether it is an element on the diagonal or otherwise. There are n squared comparisons and n squared appending.

# file : 6\_list\_identity.py

# version 2

n = 4

a = []

for i in range(1, n + 1) :

a.append([])

for j in range(1, n + 1) :

if i == j :

a[i-1].append(1)

else:

a[i-1].append(0)

for x in a :

for e in x :

print(e, end = " ")

print()

#version 3:

We avoid n \* n comparisons. We put 0 every in the matrix and then change the elements on the diagonal to 1. This has n squared appending and n assignments. This is definitely easier to understand and is efficient.

# file : 7\_list\_identity.py

# version 3

n = 4

a = []

for i in range(1, n + 1) :

a.append([])

for j in range(1, n + 1) :

a[i-1].append(0)

a[i - 1][i - 1] = 1

for x in a :

for e in x :

print(e, end = " ")

print()

**example 2:**

You may check the comments added to the program.

**display a Pascal triangle.**

# file : 8\_disp\_Pascal.py

a = [

[1],

[1, 1],

[1, 2, 1],

[1, 3, 3, 1],

[1, 4, 6, 4, 1],

[1, 5, 10, 10, 5, 1]

]

n = 5

# display Pascal triangle

#print(a)

for i in range(n + 1) : # go thro n + 1 rows from 0 to n

# output # of spaces which decreases as we move to the next row - as i increases

print(" " \* (n - i), end = "")

for j in range(i + 1): # display i + 1 elements

print("{0:6}".format(a[i][j]), end = "")

print()

$ python 8\_disp\_Pascal.py

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

This program indicates how we can control the display.

file: 9\_format.py

**# formatting**

x = 10; y = 20

print(x, y, x + y)

print("{0:5} and {1:5} is {2:6}".format(x, y, x + y))

# {0:5} output the zeroth argument of format using width of 5 characters

# {1:5} output the first argument of format using width of 5 characters

# {2:6} output the second argument of format using width of 6 characters